Digital Image Processing Using MATLAB
3rd edition

Gonzalez, Woods, and Eddins
Gatesmark Publishing

Book Website: www.ImageProcessingPlace.com

February 17, 2022

ERRATA SHEET

Some of the corrections listed may already be incorporated in your printing of the book

Page	**Reads**	**Should Read**

76	The listing for function `imageStats2` should be:
	`function G = imageStats2(f)`
	`G{1} = size(f);`
	`G{2} = mean2(f);`
	`G{3} = mean(f,2);`
	`G{4} = mean(f,1);`	

147 Ten lines from bottom	0.5%	5%
150, 3rd line of 2nd full parag.	`lp = fir1(128,0.1)`	`lp = fir1(128,0.06)`
241 [**Solution** to Proj 4.1(a)] in your Support Package	`S = complex(SG,0)`	`SG = complex(SG,0)`
242 [Proj 4.2(b)]	$MNf^*(x,y) = F^*(u,v)$	$MNf^*(x,y) = DFT[F^*(u,v)]$
242 [**Solution** to Proj 4.3(c)] in the Faculty Support Package]	`figure, imshow(g3)` `figure, imshow(g4)`	`figure, imshow(g3,[])` `figure, imshow(g4,[])`
243 [Proj 4.5(a)]	`..FrequencyEmphasis(f,a,b,D0,n)`	`..FrequencyEmphasis(f,D0,n,a,b)`
244 [Proj 4.6(b)]	. . . Fig. 4.16(b).	. . . Fig. 4.15(b).
244 [**Solution** to Proj 4.7(a)] in your Support Package	Replace lines 52 & 53 with: `CC(:,1) = r0 + delr(:);` & `CC(:,2) = c0 + delc(:);`	
316, Proj. 5.1(c), line 2	Replace “in Eq. (5-13)” with (see the 1st row in Table 5.1)	
317, Proj. 5.6(a), in the Example	. . . total of 32 squares in each size	. . . total of 32 squares in each side
317, Proj. 5.6(b), line 1	Replace “pixels in which each square has 8 pixels” by “8 squares”	
317, Proj. 5.6(b), line 2	Replace “a PSF” by “an OTF”	
374, Proj. 6.7(b)	Replace the word “resizing” by the word “reducing.”	
457	`basisImage`	`basisImages`
459, Eqs. (8-10) and (8-11)	Replace *vy/N* in the exponent of both equations by *vy\M*	
643	The 10th line should be: `gboth = edge(f,'sobel',0.10,'nothinning');`	

797	Two instances of `fchcode` should be `freemanChainCode.`	
852, middle of page	`gR{k} = P.X({:,k}`	`gR{k} = PR.X({:,k}`
857, Eq. (13-48), swap *b* & *c*	$\mathbf{H} = [a \text{ c}; c\ b]$	$\mathbf{H} = [a\ b; b\ c]$
857, Eq. (13-50)	The square root should not enclose the denominator.	
980, Proj 14.3(b)	Figs. 14.4(g)-(i)	Figs. 14.5(g)-(i)

Revised function imrecon in Proj 5.10(a)

```
function g = imrecon(f,theta)
%IMRECON Image reconstruction from projections.
%   G = IMRECON(F,THETA) creates projections of grayscale image F, then
%   reconstructs the image using backprojections at the angles supplied
%   in the 1-D array THETA. The angles in THETA are in degrees. These
%   are the angles of the normal to the direction of the beam, measured
%   counterclockwise with respect to the x-axis, as illustrated in Fig.
%   5.16. For example, to obtain a vertical projection, we use THETA = 0
%   degrees.
%
%   The output reconstructed image is square, of size equal to the long
%   dimention of the input mage. The output is scaled so the full
%   intensity range [0,1].
%
%    The objectives of this function are to illustrate conceptually the
%    basics of image reconstruction from projections, as explained in
%    Section 5.10.


% Preliminaries
% Generate a square image whose size will be the longest dimension of
% the original, as required in the function definition.
f = makeSquare(f);
M = size(f,1);
f = im2double(f);

% Pad f with a border of zeros, large enough to accomodate the largest
% possible rotated image. In general, for an image of size M x N, the
% vertical top and bottom padding are ceil((D-M)/2). For the horizontal,
% the padding the left and right padding are ceil((D-N)/2), where D is
% the diameter of the image. But our images have been padded to be
% square.
D = ceil(sqrt(M^2 + M^2));

% Make D an even integer so that padding strips will be of the same size
% on top, bottom, left, and right.
if isodd(D)
   D = D + 1;
end

% Padding value:
pad = ceil((D - M)/2);

% Pad the image
f = padarray(f,[pad,pad],0,'both');

% Beam(s) is(are) normal to angle(s) provided.
theta = theta + 90;

g = zeros(size(f));

smearLength = size(g,2);

% Projections and backprojections.

NL = numel(theta);

% A wait bar is included because this is a slow-running function for
% large images and/or a large number of angle increments.
bar = waitbar(0,'Working...');
```

```
for I = 1:NL
    % For simplicity, rotate image instead of the sensors, thus the use
    % of -theta below. This is equivalent to leaving the image stationary
    % and rotating the sensors.
    rot = imrotate(f,-theta(I),'bilinear','crop');

   % Sum rows to obtain projection.
   p = sum(rot,2);

    % "Smear" the projections across image.
    smeared = repmat(p,1,smearLength);

    % Rotate g to insert projection.
   g = imrotate(g,-theta(I),'bilinear','crop');

   % Insert projection.
   g = g + smeared;

   % Rotate back.
   g = imrotate(g,theta(I),'bilinear','crop');

   waitbar(I/NL)
end

close(bar)

% Crop back to original size.
g = g(pad+1:pad+M, pad+1:pad+M);

% Scale output to the full [0,1] range.
g = intensityScaling(g);

%-------------------------------------------------------------------%
function g = makeSquare(f)

[M,N] = size(f);
D = abs(M - N);
if isodd(D)
   D = D + 1;
end
if M > N
   padVector = [0,D/2];
elseif M < N
   padVector = [D/2,0];
else
   padVector = [0,0];
end

% Pad the image.
f = padarray(f,padVector,0,'both');

% Dimensions could be off by 1 pixel. Make sure image is square.
[M,N] = size(f);
if M ~= N && M < N
   moreRows = N  - M;
   % Make the image square by replicating rows.
   g = padarray(f,[moreRows,0],'replicate','post');
elseif M ~= N && M > N
   moreColumns = M - N;
   g = padarray(f,[0,moreColumns],'replicate','post');
else
   g = f;
end

%-------------------------------------------------------------------%
```

Revised function imHueRange in Project 7.9(a)

```
function [im,imhuerange] = imHueRangeNEW(image,angrange,type)
%IMHUERANGE Extracts angular range from HSV,HSI,HSL images.
%   [IM,IMHUERANGE] = IMHUERANGE(IMAGE,ANGRANGE,TYPE) extracts from the
%   H component of an HSV,HSI,or HSL IMAGE a range of angles specified
%   in ANGRANGE, a vector [LOW,HIGH] containing the lower and upper
%   limits of the range. Both LOW and HIGH must be whole numbers in the
%   range [0,359], with LOW <= HIGH. If ANGRANGE is a scalar in [0,359],
%   only that angular value is extracted. For example to extract
%   10-degrees on either side of yellow (60 degrees) we specify LOW =
%   50, and HIGH = 70. TYPE denotes whether the input image is 'HSV',
%   'HSI', or 'HSL'. This is a required input to protect from an
%   errorneous M-by-N-by-3 input like an RGB or CMY image.

if nargin ~= 3
   error ('Incorrect number of inputs')
elseif ~(isequal(type,'HSV') || isequal(type,'HSI') || isequal(type,'HSL'))
   error('Unknown image type')
end

% Scale the image so that its values will be in the range [0,359] which
% is the range of allowed values of angrange.
image = im2double(image)*359;

if isscalar(angrange)
   low = angrange;
   high = angrange;
else
   low = angrange(1);
   high = angrange(2);
end

% Hue component image.
H = image(:,:,1);
% Every pixel of H is an angle value. Set to zero all pixels of H whose
% values are outside the range [low,high].
imhuerange = zeros(size(H));
```

```
idx = find(H >= low & H <= high);
imhuerange(idx) = H(idx);

% Reconstruct the output image, im. This image will be of the same type
% (HSV, HSI, or HSV) as the input image, image.
im = cat(3,imhuerange,image(:,:,2),image(:,:,3)
```

Revised Solution to Project 7.9(b)

```
f = imread('dying-star-ngc6543a.tif');
figure, imshow(f);

% Input has to be HSV,HSI,or HSL
hsv = rgb2hsv(f);


% Green is at 120-degrees. Extract a range around that value.
angrange = [90 150]; % Determined experimentally.

[im,newH] = imHueRangeNEW(hsv,angrange,'HSV');

% Show the new hue compoment of the image.
figure, imshow(newH,[])

% Make a binary mask out of newH. Because function imHueReange already
% set to 0 all values outside angrange, we can use it directly to make a
% mask with values only in the range.
mask = im2double(newH > 0);

% Apply the mask to each component of hsv.
hsv = im2double(hsv);
for k = 1:3
   hsv(:,:,k) = mask.*hsv(:,:,k);
end

% Convert to rgb for display.
rgb = hsv2rgb(hsv);

% As the following image shows, the regions were extracted
% successfully. A small region in the center was also extracted because
% white contains green.
figure, imshow(rgb)
```

Revised Solution to Project 7.9(c)

```
f = imread('firebreather-midres.tif');
figure, imshow(f);

% Input has to be HSV,HSI,or HSL
hsv = rgb2hsv(f);

% Flames are between yellow (60 degrees) and red (0 degrees). Choose a
% range around 30
angrange = [22 38];

[im,newH] = imHueRange(hsv,angrange,'HSV');

% Show the new hue image.
figure, imshow(newH,[])

% Make a binary mask out of newH
mask = im2double(newH > 0);

% Apply the mask to each component of hsv.
hsv = im2double(hsv);
for k = 1:3
   hsv(:,:,k) = mask.*hsv(:,:,k);
end

% Convert to rgb.
rgb = hsv2rgb(hsv);
figure, imshow(rgb)
```